UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/542,189	04/04/2000	Gordon Taylor Davis	RAL9-2000-0008-US1	5890

25299 7590 08/04/2003

IBM CORPORATION
PO BOX 12195
DEPT 9CCA, BLDG 002
RESEARCH TRIANGLE PARK, NC 27709

EXAMINER	
HUISMAN, DAVID J	
ART UNIT	**PAPER NUMBER**
2183	9

DATE MAILED: 08/04/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/542,189	DAVIS ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *11 June 2003* .

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-31* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-31* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *04 April 2000* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)☐ The proposed drawing correction filed on _____ is: a)☐ approved b)☐ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____ .

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) ☐ The translation of the foreign language provisional application has been received.

15)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

1. Claims 1-31 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: #7. Extension of time as received on 6/11/2003 and #8. Amendment "A" as received on 6/11/2003.

Specification

3. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

Claim Objections

4. Claim 24 is objected to because of the following informalities: Insert --storage-- after "shared remote" in line 13. Also, in line 11 of claim 24, please insert --a-- before "priority". Appropriate correction is required.

5. Claim 28 recites the limitation "the thread outlets in line 9. There is insufficient antecedent basis for this limitation in the claim (the examiner recommends removing the word "the").

6. Claim 30 is objected to because of the following informalities: Replace "packet" with --thread-- in line 12. Appropriate correction is required.

Claim Rejections - 35 USC § 102

7. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in a patent granted on an application for patent by another filed in the United States before the invention thereof by the applicant for patent, or on an international application by another who has fulfilled the requirements of paragraphs (1), (2), and (4) of section 371(c) of this title before the invention thereof by the applicant for patent.

The changes made to 35 U.S.C. 102(e) by the American Inventors Protection Act of 1999 (AIPA) do not apply to the examination of this application as the application being examined was not (1) filed on or after November 29, 2000, or (2) voluntarily published under 35 U.S.C. 122(b). Therefore, this application is examined under 35 U.S.C. 102(e) prior to the amendment by the AIPA (pre-AIPA 35 U.S.C. 102(e)).

8. Claims 27, 29, and 30 are rejected under 35 U.S.C. 102(e) as being anticipated by Joy et al., U.S. Patent No. 6,341,347 B1 (herein referred to as Joy). In addition, The *American Heritage® Dictionary of the English Language, Third Edition,* 1992 (labeled DEF and herein referred to as Heritage), is cited as extrinsic evidence of the meaning of "queue/queuing" as utilized in the Joy reference.

9. Referring to claim 27, Joy has taught a thread execution control useful for the efficient execution of independent threads comprising:

a) a priority FIFO buffer for storing thread numbers. The applicant's attention should first be drawn to Heritage, which shows the general definition of queue/queuing. It can be seen that the general definition of queue is a sequence of stored data or programs awaiting processing. And, queuing would be adding data or a program to such a sequence. Joy has taught in column 4,

lines 17-27 that a particular thread may be selected based on priority. If one particular thread can be selected among a plurality of threads, then a group (i.e., a queue) of threads awaiting processing must exist from which to select. Therefore, it can be seen that threads are queued in Joy's system. Each of these threads has a corresponding thread ID number (TID), which is used to uniquely identify an active thread. See column 5, lines 4-7. Also, recall that the TID of the active thread is applied to the flip-flop system of Fig.5 in order to select the appropriate machine state for the selected thread. This TID must come from one of the threads within the group (queue) of threads, and the TID selected could be chosen based on priority, as disclosed in column 4, lines 17-25. The examiner also asserts that the applicant is claiming a priority FIFO buffer (first-in, first-out), while the applicant's disclosure shows that the queuing mechanism does not always act as a FIFO. See page 12, line 20, to page 13, line 7 of the applicant's disclosure. The thread selected from the queue is the thread with the highest priority which is making a request to execute. If the highest priority thread (at the front of the queue) is not making a request, then one of the threads behind the highest priority thread will be selected, assuming at least one of them is making a request. Therefore, applicant's FIFO is not a FIFO in that the threads do not always execute in the order they are placed in the queue. As a result, the examiner is interpreting "FIFO" as just being a general queue, which as described above, and according to the definition provided, is merely a sequence of stored data or programs awaiting execution. Consequently, the examiner asserts that Joy has taught that thread numbers are stored in a priority FIFO buffer.

b) a plurality of thread control state machines, one for each thread. See Fig.3, components 310 and 312.

c) an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machine. It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See column 4, lines 17-27. Therefore, it can be seen that the priorities of the threads within the FIFO buffer must be made available to the arbiter. In addition, a state machine is a machine with multiple states, a set of input and output events, and a means for changing states based on the input events. If the highest priority thread has been executing and it is now finished, its state machine will inherently signal to the arbiter that a next thread can be selected for execution. Otherwise, the arbiter will not know when to select a next thread.

10. Referring to claim 29, Joy has taught a thread execution control as described in claim 27. Joy has further taught that the arbiter controls the priority of execution of multiple independent threads based on the Boolean expression recited in claim 29 comprising:

a) determining whether a request R is active or inactive. From Fig.2B and column 6, lines 59-66, it can be seen that when all threads are stalled, there are no requests for execution and hence an idle processor (as denoted by reference number 240). If a thread's stall event has completed, then it is ready for execution, and it will request and make itself available for execution. Otherwise, it would continue to stall.

b) determining the priority of the threads. See column 4, lines 20-25.

c) matching the request R with the corresponding thread P. The processor will inherently determine what thread is requesting execution. For example, if three of four threads are not stalled (requesting execution control), the processor must know what three threads are available

for execution. Otherwise, in this example, the fourth thread, although not available for execution, may be matched with a request for execution from another thread, which is a violation.

d) granting a request for execution if the request is active and if the corresponding thread P has the highest priority. Again, a thread with a recently completed stall event will inherently request execution control. From the group of threads that are not stalled (requesting execution), the one with the highest priority will be chosen.

11. Referring to claim 30, Joy has taught a thread execution control as described in claim 27. Joy has further taught that the thread state control comprises control logic to:

a) dispatch a packet to a thread. It is inherent that if a thread is to execute, then it must have received some form of work to do.

b) move the thread from the initialize state to a ready state. It is inherent that after a thread has been initialized, it will be ready to execute.

c) request execution cycles for the thread. See Fig.2B and note that all threads that are executed have previously requested to be executed (hence, the insertion into the priority queue).

d) move the thread to the execute state upon grant by the arbiter of an execution cycle. See column 4, lines 17-27, and note that the arbiter will select the thread with the highest priority.

e) continue to request execution cycles while the thread is queued in the execute state. From Fig.2B, as long as the thread has not completed, it will continue to request to be executed.

f) return the thread to the initialize state if there is no latency event, or send the thread to the wait state upon occurrence of a latency event. Again, from Fig.2B, it can be seen that if a latency even occurs, then the corresponding thread will be preempted by the next thread and the stalled

thread will wait until its stall event is complete before it can be considered for execution again. If there is no latency event, and the thread finishes, the thread no longer needs to execute or wait, nor is it ready to execute, since work has not been assigned to it. Therefore, it is inherent that it should be returned to the initialization state for future execution preparation.
For applicant's benefit, this process is also shown in Fig.4 of Horvitz, U.S. Patent No. 5,752,031, which is used in succeeding rejections.

Claim Rejections - 35 USC § 103

12. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

13. Claims 1-7, 9-17, and 19-23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Callon et al., U.S. Patent No. 5,251,205 (herein referred to as Callon).

14. Referring to claim 1, Joy has taught a use of multiple threads including the steps of:

a) providing multiple instruction execution threads as independent processes in a sequential time frame. See Fig.2B and column 6, lines 59-66.

b) queuing the multiple execution threads. See column 4, lines 17-22. Note that the threads are buffered with respect to their priority. Also, although Joy has taught that the multiple execution threads have overlapping access to system resources (i.e. pipelined, see column 2, lines 44-47), Joy has not explicitly taught that the multiple execution threads have overlapping access to accessible data available in a tree search structure. However, Callon has taught the well-known

concept that router information is represented and accessed via tree search structure. See Fig.1A. It should be noted that such a tree structure is immaterial to the claimed inventive concept of thread switching and that the particulars of this structure in its claimed form have no effect on the operation of such a thread-switching system.

c) executing a first thread in the queue. See column 4, lines 20-22. The highest-priority thread that is awaiting processing will be selected for execution.

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See Fig.2B and column 6, lines 59-66.

In summation, it should be realized that the claimed inventive concept of claim 1, namely transferring execution control from a stalled thread to one of a group of queued threads, has indeed been anticipated by Joy. It is noted by the examiner that Joy does not teach the tree search structure. However, this structure has no effect on the inventive concept and is not given patentable weight in its claimed form. In addition, implementing tree data structures is well known in the art (as shown by Callon). A person of ordinary skill in the art would have recognized that by using a tree structure, inserting, deleting, and searching for data becomes more efficient in that these tasks can be performed in logarithmic time. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the system of Joy to incorporate a tree data structure in order to achieve more efficient data operations.

15. Referring to claim 2, Joy in view of Callon has taught a use of multiple threads as described in claim 1. Joy has further taught that the control of the execution is temporarily

transferred to the next thread when execution stalls due to a short latency event, and the control is returned to the original thread when the event is completed. See column 2, lines 1-3, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on a cache miss, execution control is transferred to the next highest priority thread that is waiting idle. Further note that the cache miss can be considered a short latency event since the applicant has failed to define what constitutes a short latency event. Finally, from Fig.2B, it can be seen that control is returned to the original thread after it is done stalling. Again, the claim's broad language allows for anticipation by Joy since the stall event has been completed by the time the original thread regains execution control. Also, from another viewpoint, if the original thread had highest priority in Joy's system, and if the stall event for that original thread were completed by the time the next thread encountered a stall event, then the original thread would regain control from the next thread.

16. Referring to claim 3, Joy in view of Callon has taught a use of multiple threads as described in claim 2. Joy has further taught that a processor instruction is encoded to select a short latency event. Recall that Joy has taught thread-switching when a cache-miss occurs. See column 2, lines 2-30. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data that is not in the cache, for example. Therefore, load instructions which are encoded in such a way that the processor knows to access cache, would actually select a short latency event upon a cache miss.

17. Referring to claim 4, Joy in view of Callon has taught a use of multiple threads as described in claim 1. Joy has further taught that full control of the execution is transferred to the next thread when execution of the first thread stalls due to a long latency event. See column 2,

lines 1-3, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on a cache miss, execution control is transferred to the next highest priority thread that is waiting idle. Further note that the cache miss can be considered a long latency event since the applicant has failed to define what constitutes a long latency event. Also, the cache miss would result in an access to some slower form of memory (another cache, main memory, disk), which would also be considered long latency events. Note from Fig.2B and column 6, lines 59-66, that control is given to the next thread until it encounters a stall event and the original stall event has been completed.

18. Referring to claim 5, Joy in view of Callon has taught a use of multiple threads as described in claim 4. Joy has further taught that a processor instruction is encoded to select a long latency event. Recall that Joy has taught thread-switching when a cache-miss occurs. See column 2, lines 2-30. It is inherent that such a long latency event (i.e. a cache miss and ultimately a slower memory access) would result from trying to load data that is not in the cache, for example. Therefore, load instructions which are encoded in such a way that the processor knows to access cache, would actually select a long latency event upon a cache miss.

19. Referring to claim 6, Joy in view of Callon has taught a use of multiple threads as described in claim 1. Joy has further taught queuing the threads to provide rapid distribution of access to shared memory. See Fig.3, components 330 and 320, and column 9, lines 1-5, and lines 36-40.

20. Referring to claim 7, Joy in view of Callon has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of

the coprocessor. See Fig.9 and column 19, lines 6-58. Note that the second processor (co-processor) is pipelined. Further note that the co-processor threads have overlapping access to main memory via MIU 928 as well as to all caches.

21. Referring to claim 9, Joy in view of Callon has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads are used with zero overhead to switch execution from one thread to the next. See column 6, lines 34-36, and note that the replication of registers for each thread would result in zero switching overhead, as discussed in column 4, lines 12-16.

22. Referring to claim 10, Joy in view of Callon has taught a use of multiple threads as described in claim 9. Joy has further taught that each thread is given access to general purpose registers and local data storage to enable switching with zero overhead. See column 6, lines 34-36, column 4, lines 12-16, and column 8, lines 59-67. Note that the registers are replicated for each thread and each thread is also given its own load and store buffers to hold any pending data coming from or going to memory. Furthermore, from Fig.7A, it can be seen that the data cache is divided into separate parts for each thread.

23. Referring to claim 11, it has been noted by the examiner that claim 11 claims a processor that uses multiple threads as described in claim 1. Therefore, claim 11 is rejected for the same reasons set forth in the rejection of claim 1.

24. Referring to claim 12, Joy in view of Callon has taught a processing system as described in claim 11. Furthermore, claim 12 is rejected for the same reasons set forth in the rejection of claim 2.

25. Referring to claim 13, Joy in view of Callon has taught a processing system as described in claim 12. Furthermore, claim 13 is rejected for the same reasons set forth in the rejection of claim 3.

26. Referring to claim 14, Joy in view of Callon has taught a processing system as described in claim 11. Furthermore, claim 14 is rejected for the same reasons set forth in the rejection of claim 4.

27. Referring to claim 15, Joy in view of Callon has taught a processing system as described in claim 14. Furthermore, claim 15 is rejected for the same reasons set forth in the rejection of claim 5.

28. Referring to claim 16, Joy in view of Callon has taught a processing system as described in claim 11. Furthermore, claim 16 is rejected for the same reasons set forth in the rejection of claim 6.

29. Referring to claim 17, Joy in view of Callon has taught a processing system as described in claim 16. Furthermore, claim 17 is rejected for the same reasons set forth in the rejection of claim 7.

30. Referring to claim 19, Joy in view of Callon has taught a processing system as described in claim 11. Furthermore, claim 19 is rejected for the same reasons set forth in the rejection of claim 9.

31. Referring to claim 20, Joy in view of Callon has taught a processing system as described in claim 19. Furthermore, claim 20 is rejected for the same reasons set forth in the rejection of claim 10.

32. Referring to claim 21, Joy in view of Callon has taught a processing system as described in claim 20. Joy has further taught that the local data storage is made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor. Regarding the data storage (as shown in Fig.8), Joy has taught that the cache is segregated into a first thread's portion and a second thread's portion. Fig.8 also shows the addressing format for accessing the data cache. Note that index field 812 (split into 823 and 824) includes one bit specified by the thread ID (TID), which is an identification tag unique to each thread. See column 3, lines 52-55. This thread ID, when implemented as part of the index field, provides addresses that are only accessible by the thread associated with the corresponding thread ID. The remaining bits are part of a virtual address which is supplied by the processor's load/store units. See column 8, lines 59-67. In addition, recall that there are separate registers for each thread (see column 6, lines 34-36). In Fig.13 and column 27, lines 32-37, Joy has disclosed that each plane (register window) 1310 represents a separate group of registers within a 3-dimensional register file. There is a one-to-one mapping of register windows to threads, and consequently, thread-switching also results in register window switching so that each thread has its own set of registers. Selecting a window is performed by changing the window pointer. As described in a similar fashion above, it would have been obvious to use the thread ID or a subset of the thread ID to specify which register set will be active at a given time.

33. Referring to claim 22, Joy in view of Callon has taught a processing system as described in claim 20. Joy has further taught that the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which

array will be delivered to the processor for a given thread. See Fig.13. Recall from the rejection of claim 21, that each plane 1310 represents an array of registers (register file). From Fig.13, it can be seen that the register index is decoded such that register N is selected from among each of the arrays. However, the current window pointer 1312 is decoded such that register N from the third window will be selected. Therefore, the current window pointer acts as a selector in that it selects a register window for use by the thread.

34. Referring to claim 23, Joy in view of Callon has taught a processing system as described in claim 20. Joy has further taught that the local data storage is fully addressable by the processor, an index register is contained within the register array, and the thread execution control has no address control over the local data storage or the register arrays. See Fig.8 and note the 64-bit indexing format, which, as known in the art, could be stored in one of the 64-bit registers (for indirect addressing purposes) in register file 1300 (Fig.13). One embodiment, as described in the rejection of claim 21, has part of the index field including a thread ID, which is provided based on the currently executing thread. However, according to column 18, lines 21-38, this thread ID tagging can be disabled in cases where native threads and lightweight processes are sharing the same virtual address space while being executed. This allows for a non-segregated cache, which would eliminate the under-utilization of resources resulting from certain threads.

35. Claims 8 and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Callon in view of Parady, U.S. Patent No. 5,933,627, and further in view of Horvitz, U.S. Patent No. 6,067,565.

36. Referring to claim 8, Joy in view of Callon has taught a use of multiple threads as described in claim 1.

a) Joy in view of Callon has not explicitly taught providing a separate instruction pre-fetch buffer for each execution thread. However, Parady has taught the concept of providing separate prefetch buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

b) Joy in view of Callon in view of Parady has taught providing separate prefetch buffers for each execution thread. Joy in view of Callon in view of Parady has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Horvitz has taught the concept of prefetching data during periods of low processing activity or idle processing time. A person of ordinary skill in the art would have recognized that prefetching when the processor is not fully utilized would increase the efficiency of the system. The processor would spend less time in an idle state and the prefetched data would be available ahead of time, resulting in quicker execution for the particular process (thread) using the prefetched data. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to collect

instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized.

37. Claims 24 and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Parady. In addition, The *American Heritage® Dictionary of the English Language, Third Edition,* 1992 (labeled DEF and referred to as Heritage), is cited as extrinsic evidence of the meaning of "queue/queuing" as utilized in the Joy reference.

38. Referring to claim 24, Joy has taught a processor configuration comprising:

a) a CPU with multiple threads. See Fig.2B and Fig.2C.

b) an array of general purpose registers, said array communicating with the CPU. See column 6, lines 34-36.

c) a local data storage including separate storage space for each thread. See Fig.7A and note that the data cache is divided into separate parts for each thread.

d) a thread execution control for the general register array and the local data storage. As discussed in parts (b) and (c) above, each thread has its own register set as well as its own portion of data cache. Therefore, when threads are switched, it is inherent that some type of thread execution control is used to control access to certain parts of cache and certain register files so that one thread does not interfere with another thread's resources.

e) a first coprocessor connecting the CPU to a local data storage, said thread execution control including a priority FIFO buffer to store thread numbers. See Fig.9 and note that coprocessor 902 connects the CPU 904 to a local data storage 920. Furthermore, the applicant's attention should first be drawn to Heritage, which shows the general definition of queue/queuing. It can

be seen that the general definition of queue is a sequence of stored data or programs awaiting processing. And, queuing would be adding data or a program to such a sequence. Joy has taught in column 4, lines 17-27 that a particular thread may be selected based on priority. If one particular thread can be selected among a plurality of threads, then a group (i.e., a queue) of threads awaiting processing must exist from which to select. Therefore, it can be seen that threads are queued in Joy's system. Each of these threads has a corresponding thread ID number (TID), which is used to uniquely identify an active thread. See column 5, lines 4-7. Also, recall that the TID of the active thread is applied to the flip-flop system of Fig.5 in order to select the appropriate machine state for the selected thread. This TID must come from one of the threads within the group (queue) of threads, and the TID selected could be chosen based on priority, as disclosed in column 4, lines 17-25. The examiner also asserts that the applicant is claiming a priority FIFO buffer (first-in, first-out), while the applicant's disclosure shows that the queuing mechanism does not always act as a FIFO. See page 12, line 20, to page 13, line 7 of the applicant's disclosure. The thread selected from the queue is the thread with the highest priority which is making a request to execute. If the highest priority thread (at the front of the queue) is not making a request, then one of the threads behind the highest priority thread will be selected, assuming at least one of them is making a request. Therefore, applicant's FIFO is not a FIFO in that the threads do not always execute in the order they are placed in the queue. As a result, the examiner is interpreting "FIFO" as just being a general queue, which as described above, and according to the definition provided, is merely a sequence of stored data or programs awaiting execution. Consequently, the examiner asserts that Joy has taught that thread numbers are stored in a priority FIFO buffer.

f) a shared remote storage. See Fig.9 and column 19, lines 6-58. Note that the processor(s) have access to external memory via memory interface unit (MIU) 928.

g) a pipelined coprocessor connecting the shared remote storage and the CPU. See Fig.9 and note that coprocessor 902 connects processor 904 and the shared remote storage via bus 926.

h) an instruction memory. See Fig.3, component 330. Joy has not explicitly taught a separate prefetch queue for each thread between the instruction memory and the CPU. However, Parady has taught has taught the concept of providing separate prefetch buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

39. Referring to claim 25, Joy in view of Parady has taught a processor configuration as described in claim 24. Joy has further taught that the thread execution control includes:

a plurality of thread control state machines, one for each execution thread. See Fig.3, components 310 and 312.

b) an arbiter, responsive to signals from the FIFO buffer and the state machine to determine thread execution priority. It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See column 4, lines 17-27. Therefore, it can be seen that the priorities of the threads within the FIFO buffer must be

made available to the arbiter. In addition, a state machine is a machine with multiple states, a set of input and output events, and a means for changing states based on the input events. If the highest priority thread has been executing and it is now finished, its state machine will inherently signal to the arbiter that a next thread can be selected for execution. Otherwise, the arbiter will not know when to select a next thread.

40. Claim 26 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Parady in view of Lee et al., U.S. Patent No. 5,404,560 (herein referred to as Lee), and further in view of Horvitz.

41. Referring to claim 26:

a) Joy has not explicitly taught associating each thread with a prefetch buffer. However, Parady has taught the concept of providing separate prefetch buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

b) Joy in view of Parady has not explicitly taught determining whether a buffer associated with an execution thread is full. However, Lee has taught such a concept. See column 13, lines 52-58. A person of ordinary skill in the art would have recognized that the buffer should be checked to see if it is full. By doing this, the processor can determine when it should stop fetching

instructions to fill the buffer. If the processor did not check to see if the buffer was full, then instructions would continue to be fetched and since no room is available for storage in the prefetch buffer, another instruction would have to be overwritten. Therefore, in order to ensure that buffered instructions are not overwritten, it would have been obvious to one of ordinary skill in the art at the time of the invention to determine whether a buffer associated with an execution thread is full.

c) Joy in view of Parady and further in view of Lee has not explicitly taught determining whether the thread associated with the buffer is active. However, as discussed in part (a) above, Parady has taught that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. Therefore, the system of Parady inherently determines whether the thread associated with the buffer is active. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to determine whether the thread associated with the buffer is active.

d) Joy in view of Parady and further in view of Lee has not explicitly taught that during periods that the buffer is not being used by an active execution thread, enabling the buffer to prefetch instructions for the execution thread. However, Horvitz has taught the concept of prefetching data during periods of low processing activity or idle processing time. A person of ordinary skill in the art would have recognized that prefetching when the processor is not fully utilized would increase the efficiency of the system. The processor would spend less time in an idle state and the prefetched data would be available ahead of time, resulting in quicker execution for the particular process (thread) using the prefetched data. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to collect instructions

in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized.

42. Claims 28 and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Nikhil et al., U.S. Patent No. 5,499,349 (herein referred to as Nikhil).

43. Referring to claim 28, Joy has taught a thread execution control as described in claim 27.

a) Joy has implicitly taught means for loading a thread number into the FIFO when a packet is dispatched to the processor. Recall from column 4, lines 20-25, that a single thread from a group of buffered threads is chosen, based on priority, to be the next thread to execute. In order for a processor to select a thread from among the waiting threads, there must be a means for loading each thread into the buffer when the thread is assigned work to do. It is inherent that the thread number must be enqueued so that the processor can choose the correct thread and schedule it when necessary. If the thread number was not inserted into the scheduler, then it would not be available for scheduling

b) Joy has also implicitly taught means for unloading a thread number from the FIFO when a packet has been enqueued for transmission. When a thread has completed, it would inherently be removed from the queue, signifying that it no longer needs to be executed.

c) Joy has not explicitly taught a sole priority scheme. Instead, Joy has suggested that different priority schemes may be used when dealing with the scheduling of threads. See column 4, lines 20-25. Furthermore, Nikhil has taught a standard FIFO priority scheme such that the first thread buffered is the first thread scheduled, as is known in the art. See column 1, lines 51-55. A person of ordinary skill in the art would have recognized that if a thread stalls in Joy's system, the next thread to be executed would be at the front of the queue (i.e. highest priority). In a

FIFO, enqueuing a stalling thread would result in the thread being placed at the back of the queue behind every other thread (i.e. lowest priority). This is the nature of a FIFO. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement a FIFO thread scheduling scheme in Joy's system, wherein when a long latency event occurs, a thread number is transferred from highest priority to lowest priority in the FIFO.

d) As described in part (c), it would have been obvious to implement a FIFO thread scheduling algorithm in Joy's system. It would then follow, due to the nature of the FIFO, that the thread outlets of the FIFO used to determine priority depending on the length of time a thread has been in FIFO. For instance, in a FIFO, the highest priority thread would be the thread at the front, which would also be the thread that has been waiting the longest (since new threads are added in the back).

44. Referring to claim 31, Joy has taught a thread executing control according to claim 28. Joy has further taught that the FIFO (queue of threads) further includes means to detect occurrence of latency events. See column 7, lines 1-4. Note that if a latency event occurs, the FIFO (queue) responds by providing a next thread for execution.

Response to Arguments

45. Applicant's arguments filed on June 11, 2003, have been fully considered but they are not persuasive.

46. In the remarks, Applicant argues the novelty of claims 27, 29, and 30 on pages 11-12, in substance that:

> "...nowhere in the Joy reference is a teaching of a) a priority FIFO for storing thread numbers and c) an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machine."

47. These arguments are not found persuasive for the following reasons:

a) The applicant's attention should first be drawn to an extrinsic reference (labeled DEF), which shows the general definition of queue/queuing. It can be seen that the general definition of queue is a sequence of stored data or programs awaiting processing. And, queuing would be adding data or a program to such a sequence. Joy has taught in column 4, lines 17-27 that a particular thread may be selected based on priority. If one particular thread can be selected among a plurality of threads, then a group (i.e., a queue) of threads awaiting processing must exist from which to select. Therefore, it can be seen that threads are queued in Joy's system. Each of these threads has a corresponding thread ID number (TID), which is used to uniquely identify an active thread. See column 5, lines 4-7. Also, recall that the TID of the active thread is applied to the flip-flop system of Fig.5 in order to select the appropriate machine state for the selected thread. This TID must come from one of the threads within the group (queue) of threads, and the TID selected could be chosen based on priority, as disclosed in column 4, lines 17-25. The examiner also asserts that the applicant is claiming a priority FIFO buffer (first-in, first-out), while the applicant's disclosure shows that the queuing mechanism does not always act as a FIFO. See page 12, line 20, to page 13, line 7 of the applicant's disclosure. The thread selected from the queue is the thread with the highest priority which is making a request to execute. If the highest priority thread (at the front of the queue) is not making a request, then one of the threads behind the highest priority thread will be selected, assuming at least one of them is making a request. Therefore, applicant's FIFO is not a FIFO in that the threads do not

always execute in the order they are placed in the queue. As a result, the examiner is interpreting "FIFO" as just being a general queue, which as described above, and according to the definition provided, is merely a sequence of stored data or programs awaiting execution. Consequently, the examiner asserts that Joy has taught that thread numbers are stored in a priority FIFO buffer.

b) It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See column 4, lines 17-27. Therefore, it can be seen that the priorities of the threads within the FIFO buffer must be made available to the arbiter. In addition, a state machine is a machine with multiple states, a set of input and output events, and a means for changing states based on the input events. If the highest priority thread has been executing and it is now finished, its state machine will inherently signal to the arbiter that a next thread can be selected for execution. Otherwise, the arbiter will not know when to select a next thread.

48. In the remarks, Applicant argues the rejection of claims 1-7, 9-17, and 19-22 on pages 12-14, in substance that:

> "...no reasonable interpretation of any language in the reference or the one particular set forth in column 4, lines 17-22 could ever construe that that teaching covers queuing."
>
> "...Callon in general and Fig.1A in particular does not show tree structure for data as the examiner seems to suggest."

49. These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, the applicant's attention should first be drawn to an extrinsic reference (labeled DEF) which shows the general definition of queue/queuing. It can be seen that the general definition of queue is a sequence of stored data or programs awaiting processing.

And, queuing would be adding data or a program to such a sequence. Joy has taught in column 4, lines 17-27 (as set forth in the previous office action) that a particular thread may be selected based on priority. If one particular thread can be selected among a plurality of threads, then a group (i.e., a queue) of threads awaiting processing must exist from which to select. Therefore, it can be seen that threads are queued in Joy's system. The applicant further argues on page 13 that flip-flops are used to switch threads. It is noted by the examiner that these flip-flops are used to switch machine states as disclosed in column 3, lines 35-40. However, the examiner asserts that this system (the flip-flop system shown in Fig.5) is merely reactive to a thread being selected. It can be seen from Fig.5 and column 13, lines 34-43, that the flip-flop system is activated by a thread identifier signal (TID), which selects a particular thread. However, the TID must have been previously generated from within the system, where the TID corresponds to one of "X" amount of threads within a group (or queue) of threads ready for execution. As a result, it can be seen that X amount of states can be achieved by the flip-flop system due to the existence of X amount of TIDs, which in turn means that multiple threads must be queued for execution.
b) Regarding the second argument, the applicant is correct in stating that Fig.1A does not show the data structure. However, the examiner had made a typographical error in listing Fig.1A as the figure which anticipates the corresponding portion of claim 1. Instead, the examiner had meant to reference Fig.4B, which shows a tree with node data structures, which are in turn shown in Fig.5A. These are the figures that are relevant to claim 1.

50. In the remarks, Applicant argues the rejection of claims 8 and 18 on pages 14-15, in substance that:

> "...claims 8 and 18 are combination claims depending upon claims that require queuing. As argued above, the examiner's base references, Joy and Callon, do not teach queuing. The newly cited references, Parady and Horvitz, do not address queuing. Therefore, claims 8 and 18 are patentable over the art of record."
>
> "...claims 8 and 18 are patentably distinct because none of the references selectively load separate instruction prefetch buffers as recited in the claims."

51. These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, it has been explained above by the examiner that Joy does teach queuing. Therefore, claims 8 and 18 are not patentable over the prior art of record when referring to queuing.

b) Regarding the second argument, Parady has shown separate prefetch buffers for each thread and Horvitz has been used to show that prefetching can be done during periods of low processing activity or idle processing time, which reads on claims 8 and 18 since they are claiming prefetching when threads are idle.

52. In the remarks, Applicant argues the rejection of claim 26 on pages 15-16, in substance that:

> "...in order for references to be combined to render a claim obvious the motivation should be in at least one of the references or the examiner give logical and concrete reasons why the combination would have been obvious to one skilled in the art. By the examiner's own argument it appears that none of the references set forth the motivation. The examiner's argument does not set forth any logical or concrete reason why an artisan viewing these references would form the combination."

53. These arguments are not found persuasive for the following reasons:

a) The applicant's attention should be drawn to the following:

The test of obviousness is:

> "whether the teachings of the prior art, taken as a whole, would have made obvious the

> claimed invention," *In re Gorman*, 933 F.2d at 986, 18 USPQ2d at 1888.
>
> Subject matter is unpatentable under section 103 if it "'would have been obvious . . . to a person having ordinary skill in the art.' While there must be some teaching, reason, suggestion, or motivation to combine existing elements to produce the claimed device, it is not necessary that the cited references or prior art specifically suggest making the combination." *In re Nilssen*, 851 F.2d 1401, 1403, 7 USPQ2d 1500, 1502 (Fed. Cir. 1988).
>
> "Such suggestion or motivation to combine prior art teachings can derive solely from the existence of a teaching, which one of ordinary skill in the art would be presumed to know, and the use of that teaching to solve the same [or] similar problem which it addresses." *In re Wood*, 599 F.2d 1032, 1037, 202 USPQ 171, 174 (CCPA 1979).
>
> "In sum, it is off the mark for litigants to argue, as many do, that an invention cannot be held to have been obvious unless a suggestion to combine prior art teachings is found *in* a specific reference."

Entire quote from *In re Oetiker*, 24 USPQ2d 1443 (CAFC 1992).

In summation, the combined references are not required to disclose or specifically suggest particular elements. Instead, the measure is what the teachings of the references would suggest to one of ordinary skill in the art. The applicant should note from the previous office action, pages 17-18 (parts a-d) that the examiner has set forth logical reasons as to why one of ordinary skilled in the art would combine to references in question. As a result, claim 26 is rendered obvious.

54. In the remarks, Applicant argues the rejection of claim 28 on pages 16-17, in substance that:

> "...the examiner's argument is based upon hindsight and information gleaned from the applicant's invention disclosure. For example, the examiner on page 19, item b) of the office action states: 'Joy has also implicitly taught means for unloading a thread number from the FIFO when a packet has been enqueued for transmission.' There is no such implicit teaching in the Joy reference."
>
> "...Nikhil does not provide the deficiency which is present in the Joy reference. Therefore, the examiner's combination does not render claim 28 obvious."

55. These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, the examiner asserts that hindsight was not used. Instead, recall from above that the examiner has argues that Joy has taught a queue of threads waiting to be executed. Each of these threads has a corresponding thread ID number (TID), which is used to uniquely identify an active thread. See column 5, lines 4-7. Also, recall that the TID of the active thread is applied to the flip-flop system of Fig.5 in order to select the appropriate machine state for the selected thread. This TID must come from one of the threads within the group (queue) of threads, and the TID selected could be chosen based on priority, as disclosed in column 4, lines 17-25. The examiner also asserts that the applicant is claiming a FIFO (first-in, first-out), while the applicant's disclosure shows that the queuing mechanism does not always act as a FIFO. See page 12, line 20, to page 13, line 7. The thread selected from the queue is the thread with the highest priority which is making a request to execute. If the highest priority thread (at the front of the queue) is not making a request, then one of the threads behind the highest priority thread will be selected, assuming at least one of them is making a request. Therefore, applicant's FIFO is not a FIFO in that the threads do not always execute in the order they are placed in the queue. As a result, the examiner is interpreting "FIFO" as just being a general queue, which as described above, and according to the definition provided, is merely a sequence of stored data or programs awaiting execution. Consequently, the examiner asserts that

Joy has implicitly taught that thread numbers are unloaded from a queue when a particular thread is to execute (i.e., a packet has been enqueued for transmission).

b) Regarding the second argument, as described in part a) above, the applicant's queue is not acting strictly as a FIFO, and therefore, Joy does not contain the deficiency (claim 28(a)) mentioned by applicant. Consequently, Nikhil does not need to teach this portion of the claim.

Conclusion

56. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (703) 305-7811. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (703) 305-9712. The fax phone numbers for the

organization where this application or proceeding is assigned are (703) 746-7239 for regular communications and (703) 746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.

RICHARD L. ELLIS
PRIMARY EXAMINER

DJH
David J. Huisman
July 31, 2003

DEF

queue

queue (kyo͞o) *noun*

1. A line of waiting people or vehicles.
2. A long braid of hair worn hanging down the back of the neck; a pigtail.
3. *Computer Science.* A sequence of stored data or programs awaiting processing.

verb, intransitive

queued, queu·ing, queues

To get in line: *queue up at the box office.*

[French, from Old French *cue*, tail, from Latin *cauda, cōda.*]